Behavioral Cloning

Behavioral Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network that can drive autonomously on track one at max speed
- model2.h5 containing a trained convolution neural network that can drive autonomously on both tracks on moderate speed.
- writeup_report.md or writeup_report.pdf summarizing the results

2. Submission includes functional code

Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing

```
python drive.py model.h5
```

There is an additional model saved as model2.h5. This can be used to drive the car around both tracks at moderate speed (15 mph)

3. Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of a convolution neural network with 5x5 and 3x3 filter sizes and depths between 24 and 64 (model.py lines 31-59). This was based on the NVIDIA PilotNet architecture.

The model includes ELU layers to introduce nonlinearity (code lines 36, 38, 40, 42, 44, 48, 50 and 52), and the data is normalized in the model using a Keras lambda layer (code line 34).

While it's common to use no activation function at the output node of a regression model, I have used 'tanh' activation function at the output. tanh function is particularly suitable for regressions with target variable between -1.0 and 1.0, with higher density around 0.0 which is the case for this particular application.

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce over fitting (model.py line 46). In addition to that I have also used L2 regularization as well.

The model was trained and validated on different data sets to ensure that the model was not over fitting (code line 132). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not required to be tuned manually. A suitable value of 0.0001 was used (model.py line 56). Similarly dropout of 0.2 and l2 beta of 1e-6 was selected by trial and error. One thing should note here is that, while dropout and L2 regularization helps to prevent over fitting a model, larger values of these seem to cause model to regression into average of training data instead of converging to model features.

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road and driving in the opposite direction of the track.

For details about how I created the training data, see the next section.

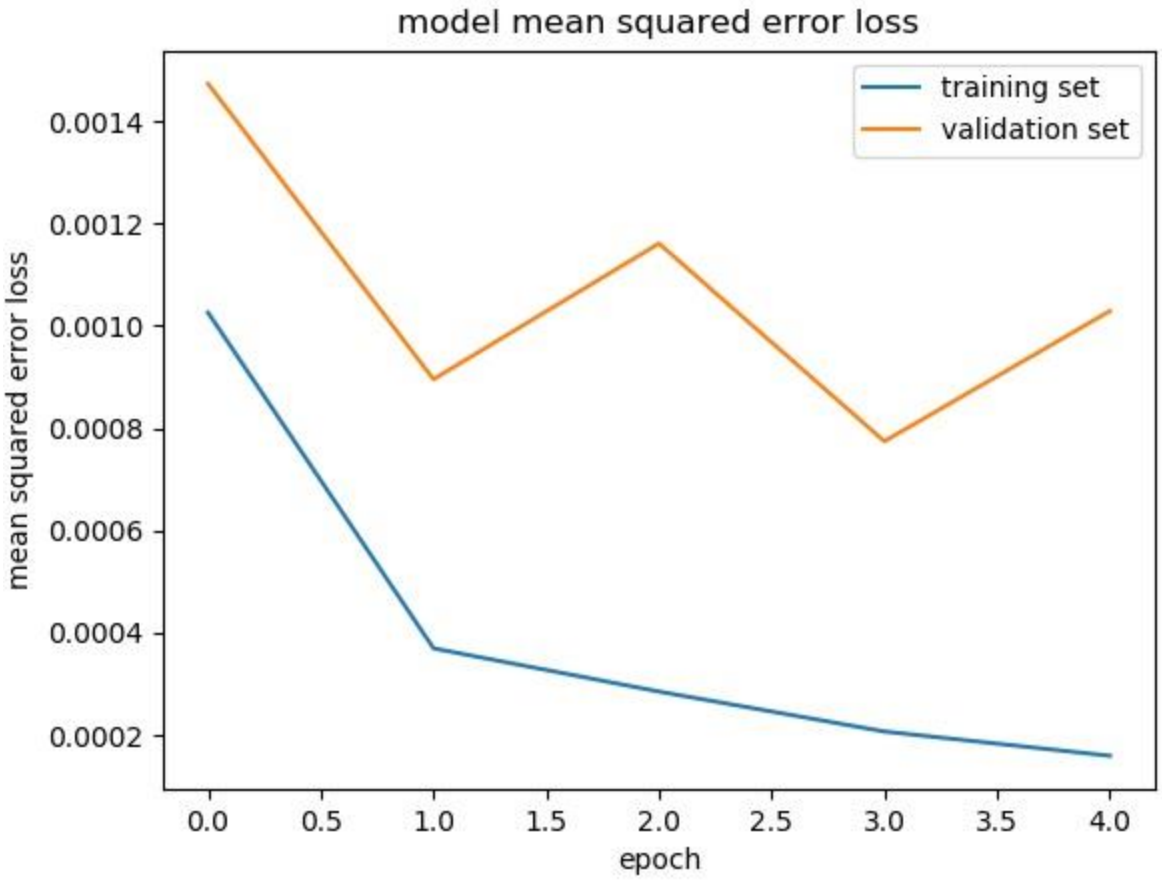
Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to iteratively refine a proven model suitable for the task.

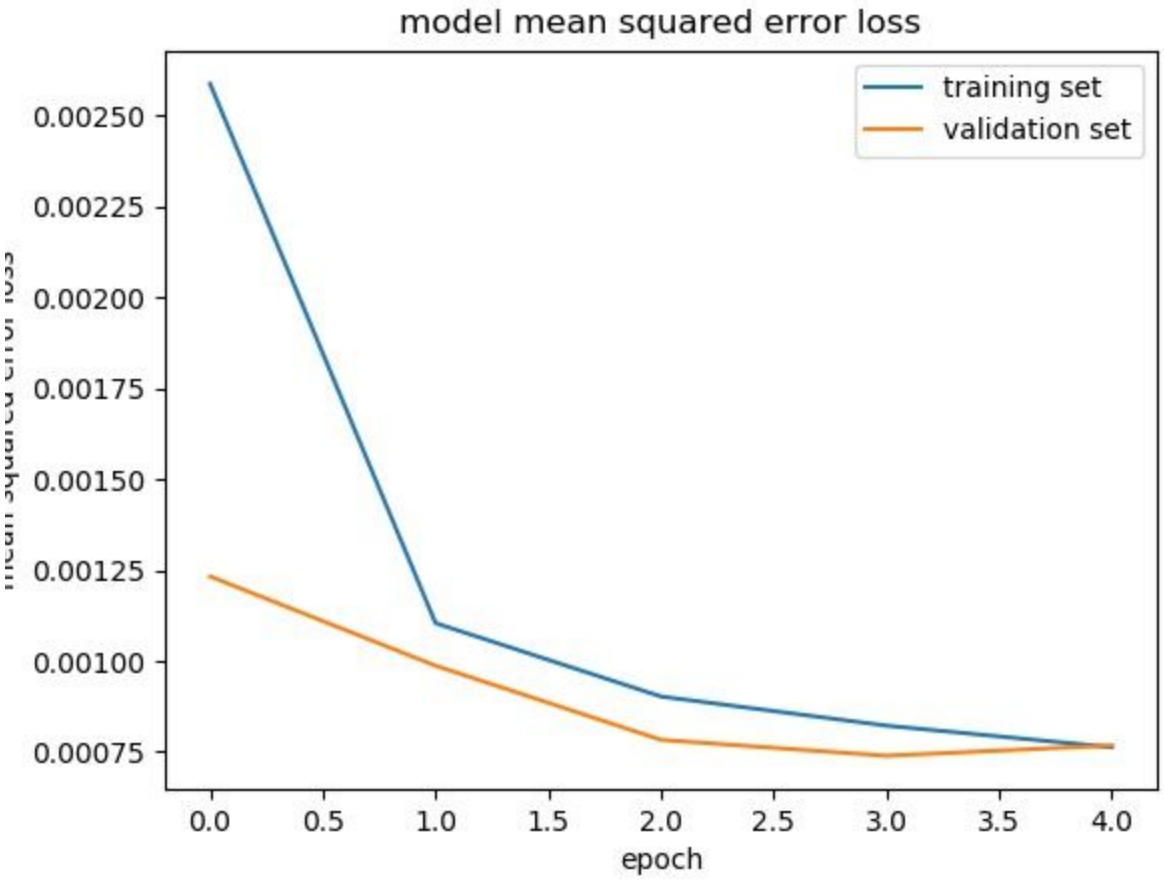
My first step was to use a convolution neural network model similar to the NVIDIA PilotNet architecture. I thought this model might be appropriate because it was successful in real world application of end-to-end learning of self-driving cars.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was over fitting.



To combat the over fitting, I modified the model so that it includes a Dropout layer after the Flatten layer and also introduced L2 regularization as well.

Then I shuffled the input data before training to ensure that training and validation set are equal in distribution. These steps prevented model from over fitting and started to generalize on data set as evident by training and validation loss as below.



The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track. to improve the driving behavior in these cases, I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to recover if it drifts toward the side of the road. To augment this, I also included left and right camera images of the data set with a suitable correction value applied to corresponding steering angle. Finally I have also recorded one additional lap driving in the opposite direction of the track.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 18-24) consisted of a convolution neural network with the following layers and layer sizes. This was based on the NVIDIA PilotNet architecture.

Layer	Description
Input	66x200x3 YUV image
Normalization	x/127.5 - 1.0
Convolution 5x5	2x2 stride, outputs 31x98x24
ELU	
Convolution 5x5	2x2 stride, outputs 14x47x36
ELU	
Convolution 5x5	2x2 stride, outputs 5x22x48
ELU	
Convolution 3x3	1x1 stride, outputs 3x20x64
ELU	
Convolution 3x3	1x1 stride, outputs 1x18x64
ELU	
Flatten	1152 neurons
Dropout	Dropout probability: 0.2
Fully connected	100 output units
ELU	
Fully connected	50 output units
ELU	
Fully connected	10 output units
ELU	
Fully connected	1 output units
tanh	

Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)

(Image source: https://devblogs.nvidia.com/parallelforall/wp-content/uploads/2016/08/cnn-architecture-624x890.png)

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to recover if it drifts towards the side of the road. These images show what a recovery looks like starting from left side of the road:

Then I recorded another lap on track one while driving in the opposite direction of the track.

Then I repeated this process on track two in order to get more data points. However given that track two has lane divider in middle I wanted to mimic real driving behavior by driving only on the right hand side lane of the track. To be safe from falling off track I stayed closed to the center of the track but tried my best not to cross the lane divider. (Except for occasional small mistakes on curves) Following image show how driving on the second track looks like.

This data set can be downloaded from: https://s3-us-west-2.amazonaws.com/aravindadp/sdcnd/data.zip

To augment the data set, I also flipped images and angles thinking that this would enable the model to generalize and also prevent biasing towards the left curves on track one. For example, here is an image that has then been flipped:

Image captured by the center camera

Flipped image from the center camera

However I did not augment the data in track two by flipping the image since my intention was to drive along the right side lane of the track and this represents asymmetric driving behavior.

After the collection process, I had 47970 and 49077 in respective tracks for a total of 97047 number of data points. I then preprocessed this data by converting to YUV color space and cropping 55 and 25 pixels respectively from top and bottom of the image. Finally I rescaled these images to 200 pixel by 66 pixel images to match the input size of NVIDIA model.

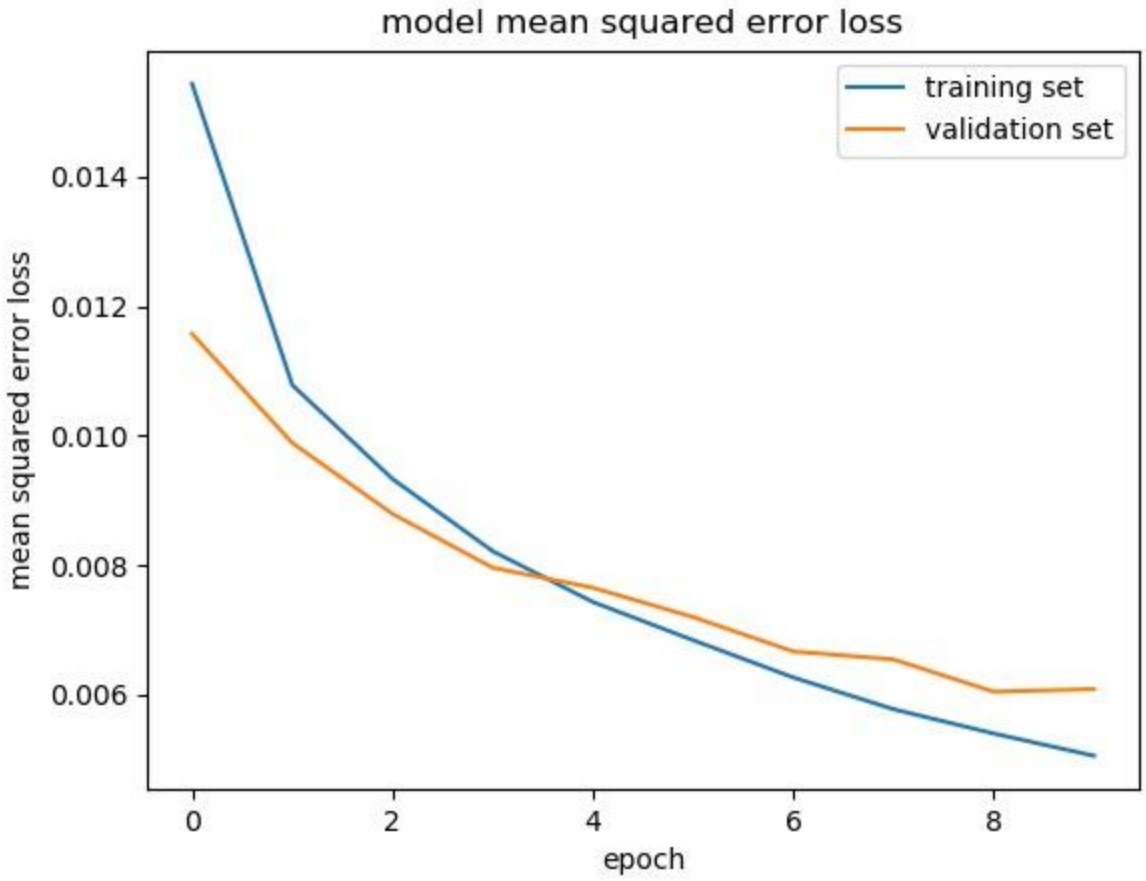
Following image show how the input image looks like after cropping. (In RGB for brevity in visualizing)

Original image taken from the simulator

Cropped image

I finally randomly shuffled these data sets and put 20% of the data into a validation set.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 4 to 5 as evidenced by training and validation loss. I used an adam optimizer so that manually training the learning rate wasn't necessary.



For the training I first trained the model with data set from track one (code lines 164-167). This model was saved as model.h5. This model was capable of achieving maximum speed on track one. Autonomously driving on track one at max speed using this model was saved as video.mp4

To better generalize model further I then loaded this model and reduced Dropout to 0.1. On this pre-trained model I trained samples from track two for 2 epochs to fine tune the model for any additional features available in track two (code lines 176-198).

Then I had freeze the weights from first 3 convolutional layers and reset the weights of fully connected layers to their corresponding initial weights from untrained state of model.h5 (code lines 206-210). Here I observed that initial weights of these layers were only adjusted marginally in most cases. So randomly initializing the weights of these layers caused network to under fit. To prevent this I had set the weights of these layers from initial weights of untrained model.h5 (Which I had previously saved as model_init.h5 before training model.h5)

On this model I trained the entire set of training samples obtained from both tracks for 5 epochs. Then one more epoch on only track one to obtain model2.h5 (code lines 213-232). This model was capable of handling both tracks at a moderate speed of 15 mph. Autonomously driving on both tracks at 15 mph using this model was saved as run2_track1.mp4 and run2_track2.mp4 respectively.